Introdução

A Programação Orientada a Objetos (POO) é um dos paradigmas mais utilizados no desenvolvimento de software moderno, devido à sua capacidade de aproximar o modelo computacional da realidade. Com a POO, é possível criar sistemas mais robustos, modulares e reutilizáveis, o que facilita tanto o desenvolvimento quanto a manutenção. No projeto desenvolvido, que envolve o gerenciamento de uma biblioteca, utilizamos os pilares da POO para construir um sistema eficiente e bem estruturado. Este projeto implementa funcionalidades como empréstimo de livros, cadastro de usuários e controle de estoque, baseando-se nos princípios de encapsulamento, herança, polimorfismo e abstração.

Este trabalho analisa como esses quatro pilares foram aplicados no projeto, destacando os principais trechos de código que ilustram cada conceito.

Encapsulamento

O encapsulamento é um mecanismo que visa proteger os atributos internos de uma classe, permitindo que o acesso e a manipulação desses dados ocorram apenas por meio de métodos controlados. No projeto, aplicamos o encapsulamento principalmente nas classes `Livro` e `Usuario`. Atributos como `Titulo`, `Autor` e `QuantidadeEmEstoque`, na classe `Livro`, estão protegidos, sendo acessíveis apenas por meio de métodos `get` e `set`.

Por exemplo, a classe `Livro` possui o seguinte código que exemplifica o uso do encapsulamento:

```
public class Livro
{
    public string Titulo { get; set; }
    public string Autor { get; set; }
    private int QuantidadeEmEstoque { get; set; }
}
```

Aqui, `QuantidadeEmEstoque` é um atributo privado, o que significa que ele não pode ser modificado diretamente de fora da classe. Somente métodos da própria classe podem alterar seu valor, garantindo integridade e controle sobre o estado do objeto.

Herança

A herança é o princípio que permite criar novas classes baseadas em classes existentes, facilitando a reutilização de código. Em nosso projeto, a classe `Livro` herda de uma classe abstrata chamada `ItemBiblioteca`, que define propriedades e métodos comuns a todos os itens da biblioteca.

Abaixo está um exemplo da aplicação da herança no projeto:

```
public abstract class ItemBiblioteca
{
    public string Titulo { get; set; }
    public string Codigo { get; set; }
    public abstract void Emprestar(Usuario usuario);
    public abstract void Devolver();
}

public class Livro : ItemBiblioteca
{
    public string Autor { get; set; }
    public override void Emprestar(Usuario usuario) { ... }
    public override void Devolver() { ... }
}
```

Ao herdar de `ItemBiblioteca`, a classe `Livro` reutiliza propriedades como `Titulo` e `Codigo` e implementa os métodos abstratos `Emprestar` e `Devolver`, com sua lógica específica para livros.

Polimorfismo

O polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme, desde que sigam o mesmo contrato, seja por herança ou interfaces. No projeto, implementamos o polimorfismo por meio da sobrescrita de métodos, permitindo que a chamada de `Emprestar` e `Devolver` funcione de maneira polimórfica.

Por exemplo, na classe `Livro`, o método `Emprestar` é sobrescrito para tratar a lógica específica de empréstimo de livros:

```
public override void Emprestar(Usuario usuario)
{
    if (QuantidadeEmEstoque > 0)
    {
        QuantidadeEmEstoque--;
        Console.WriteLine($"Livro '{Titulo}' emprestado a
{usuario.Nome}.");
    }
    else
    {
        Console.WriteLine("Não há exemplares disponíveis.");
    }
```

```
}
```

Com o polimorfismo, o sistema pode tratar todos os itens da biblioteca de maneira uniforme, mesmo que cada classe tenha suas particularidades no comportamento dos métodos `Emprestar` e `Devolver`.

Abstração

A abstração é um conceito que permite que o desenvolvedor se concentre nos aspectos essenciais do objeto, ocultando detalhes desnecessários. Em nosso projeto, utilizamos tanto classes abstratas quanto interfaces para definir contratos que as classes devem seguir. A classe `ItemBiblioteca`, por exemplo, é uma abstração de qualquer item que possa ser emprestado na biblioteca.

Além disso, a interface `IEmprestavel` define os métodos obrigatórios para os itens que podem ser emprestados, garantindo que todos os itens que implementem essa interface terão os métodos `Emprestar` e `Devolver`:

```
public interface IEmprestavel
{
    void Emprestar(Usuario usuario);
    void Devolver();
}
```

Isso permite que qualquer tipo de item que herde de `ItemBiblioteca` ou implemente a interface `IEmprestavel` seja tratado de maneira genérica no sistema, abstraindo os detalhes específicos de cada tipo de item.

Conclusão

A implementação deste projeto proporcionou um aprendizado prático sobre os pilares da Programação Orientada a Objetos. Através do uso de encapsulamento, herança, polimorfismo e abstração, foi possível construir um sistema modular, reutilizável e flexível. Esses pilares não só facilitaram a organização do código, mas também promoveram a clareza e a escalabilidade do sistema. O desenvolvimento do projeto consolidou os conceitos teóricos da POO e mostrou sua aplicação prática em um cenário realista.

Referências

Java Guia do Programador